ASSIGNMENT 3

Handout: **Tuesday, 25 October 2016**
Due: **11:30 am, Thursday, 3 November 2016**

GOALS:

- Understand better the importance of information hiding;
- Design and implement Java classes;
- Get used to the IntelliJ Idea IDE;

1. EMPLOYEE AND MANAGER

Read `SalaryLevel.java`, `Employee.java`, and `Manager.java` files in the assignment project, then add the missing code to meet the requirement in `Manager.main`.

WHAT TO DO:

`SalaryLevel.java` (Task 1 and 2): add necessary code to enum `SalaryLevel` and its method `getScale`, so that `ENTRY.getScale()` returns 1, `JUNIOR.getScale()` returns 1.3, and so on;

`Employee.java` (Task 3 and 4): add necessary code to initialize an employee and calculate his/her salary;

`Manager.java` (Task 5 and 6): add necessary code to initialize a manager and calculate his/her salary.

WHAT TO HAND IN:

`SalaryLevel.java` , `Employee.java` , and `Manager.java`

2. Polymorphism and Dynamic Binding

Suppose we have three classes `Hero`, `Warrior`, and `Healer` as defined below.

```
abstract class Hero{

    private String name;
    private int level;
    private int health;

    public Hero(String s){
        name = s;
        level = 1;
        health = 100;
    }

    public String getName() {
        return name;
    }

    public int getLevel() {
        return level;
    }

    public int getHealth() {
        return health;
    }

    public void setHealth(int health){
        this.health = health;
        if(health == 0){
            System.out.println(name + " is dead.");
        }
    }

    public abstract void doAction(Hero other);

    public void levelUp(){
        level++;
        setHealth(100);
    }

}

class Warrior extends Hero{

    public Warrior(String name){
        super(name);
    }

    public void doAction(Hero other){
```

```
        int damage = Math.min(getLevel() * 5, other.getHealth());
        other.setHealth(other.getHealth() - damage);
        System.out.println(getName() + " attacks " + other.getName() + ". Does " + damage + " damage.");
    }

    public void levelUp(){
        super.levelUp();
        System.out.println(getName() + " is now a level " + getLevel() + " warrior.");
    }
}

class Healer extends Hero{

    private int mana;

    public Healer(String name){
        super(name);
        mana = 100;
    }

    public void doAction(Hero other){
        if(mana >= 10){
            int h = Math.min(getLevel() * 10, 100 - other.getHealth());
            other.setHealth(other.getHealth() + h);
            mana -= 10;
            System.out.println(getName() + " heals " + other.getName() + " by " + h + " points.");
        }
    }

    public void levelUp(){
        super.levelUp();
        mana = 100;
        System.out.println(getName() + " is now a level " + getLevel() + " healer.");
    }
}
```

Given the following variable declarations:

```
Hero hero;
Warrior warrior;
Healer healer;
```

Indicate, for each of the code fragment below, whether it compiles. If the code fragment does not compile, explain why this is the case. If the code fragment compiles, specify the text that is printed to the screen when the code fragment is executed. This is a pen-and-paper task; you are not supposed to use an IDE.

Example:

```
warrior = new Warrior();
```

```
warrior.levelUp();
```

This code does not compile, because default constructor is not available for class `Warrior`.

Task 1:

```
warrior = new Warrior("Thor");
warrior.levelUp();
```

Compile!
Result: Thor is now a level 2 warrior.

Task 2:

```
hero = new Hero("Althea");
hero.levelUp();
```

Doesn't Compile!
Reason: Hero is abstract class, cannot be initialized.

Task 3:

```
warrior = new Warrior("Thor");
healer = new Healer("Althea");
warrior.doAction(healer);
```

Compile!
Result: Thor attacks Althea. Does 5 damage.

Task 4:

```
warrior = new Healer("Diana");
warrior.levelUp();
```

Doesn't Compile!
Reason: Healer class doesn't match Warrior class

Task 5:

```
hero = new Warrior("Thor");
hero.doAction(hero);
hero = new Healer("Althea");
hero.doAction(hero);
```

Compile!
Result: Thor attacks Thor. Does 5 damage.
Althea heals Althea by 0 points.

Task 6:

```
hero = new Warrior("Thor");
warrior = hero;
warrior.doAction(hero);
```

Doesn't Compile!
Result: Warrior is a subclass of Hero. Static type of reference warrior is Warrior and it of hero is Hero, which doesn't match. Subclass object could be copied by a superclass object but this cannot be done in a reverse way.

WHAT TO HAND IN:

Your answers for the code fragments above.